

Copyright
by
Matthew Caldwell Slowik
2013

**The Report Committee for Matthew Caldwell Slowik Certifies that this is the
approved version of the following report:**

A Flexible Display System for Embedded Applications

Committee:

Adnan Aziz, Supervisor

Andreas Gerstlauer

A Flexible Display System for Embedded Applications

by

Matthew Caldwell Slowik B.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2013

A Flexible Display System for Embedded Applications

Matthew Caldwell Slowik M.S.E

The University of Texas at Austin, 2013

Supervisor: Adnan Aziz

Flat electronic displays are ubiquitous in our environment from small handheld devices to large HDTVs. This report documents the design and development of a flexible electronic display system for low cost embedded applications. The main use of such a device would be in functions where flat displays are too rigid and too brittle to operate, like the emerging wearable electronics sector. A 10cm x 15cm, 224 pixel LED mesh display system is presented as a solution. A complete software stack was created in conjunction with the hardware. On the software side, a Visual Basic application allows users to easily generate image content for the display, five different global motion algorithms permit real time image manipulation, and an AVR assembly procedure handles screen refreshing. Altogether the system weighs 176 grams, can function under normal operating conditions for eight hours before recharging is required, and costs under \$85 without volume discounts.

Table of Contents

List of Figures	vii
<i>Introduction</i>	1
Chapter 1: <i>System Design Overview</i>	3
1.1 Background	3
1.2 Flexible Display Target Specifications	4
Chapter 2: <i>Hardware Design</i>	9
2.1 Hardware Architecture Overview	9
2.2 Flexible Display Construction	12
2.3 LED Pixel Controller	17
2.4 Power Delivery	19
Chapter 3: <i>Software Design</i>	20
3.1 Software Architecture Overview	20
3.2 Image Translator User Application	21
3.3 System Control Software	23
3.4 Display Device Driver	26
Chapter 4: <i>Results</i>	31
4.1 Power Analysis	31
4.2 Power Supply Infrastructure	39
4.3 Cost Analysis	40
4.4 Weight Analysis	41
4.5 Summary of Results	42
Chapter 5: <i>Conclusions</i>	44
5.1 Summary of Key Contributions.....	44

5.2 Future Work	44
5.3 Related Technologies	46
Appendix: <i>Links to Code and Detailed Measurements</i>	48
Power Analysis Backup Data	48
User Application Code	48
System Control Software Code	48
Display Device Driver Code	48
References	49

List of Figures

Figure 1: System Level Design Parameters and Targets	5
Figure 2: Flexible Display Concept Application	6
Figure 3: Global Motion Examples	8
Figure 4: Hardware System Integration Overview	9
Figure 5: Atmel Device Memory Comparison	11
Figure 6: ATmega328 Maximum Frequency vs. Supply Voltage	11
Figure 7: WS2811 Logical Connectivity Example	12
Figure 8: Integrated LED+WS2811 Reel Segment	13
Figure 9: Horizontal Abutment of Screen Columns	14
Figure 10: Integrated LED+WS2811 Segment Side View	16
Figure 11: Integrated LED+WS2811 Folded Technique, Top View	16
Figure 12: WS2811 Handshaking Protocols	18
Figure 13: System Software Interaction	20
Figure 14: VBA Image Translator User Application	21
Figure 15: Memory Map of Image Content	25
Figure 16: REFRESH_SCREEN Byte Loop Pseudo Code and Timing	27
Figure 17: Scope Readings of WS2811 '0' Codes	29
Figure 18: Scope Readings of WS2811 '1' Codes	30
Figure 19: 0.25x Pixel Usage Color Variant Images	33
Figure 20: System Current verses User Mode with 0.25x Display Usage	34
Figure 21: Images Used to Determine an Upper Bound on Total System Current.....	35
Figure 22: System Current verses Pixel Count (White Light Only)	36
Figure 23: System Current Envelope verses Pixel Count	37
Figure 24: Example 64-pixel Images	38
Figure 25: Example 64-pixel Image Implementations	39
Figure 26: Overall System Component Material Cost	41
Figure 27: Overall System Component Weight	42

Figure 28: Summary of Results	43
Figure 29: Flexible Display Resolution Improvement	45

Introduction

Electrical display systems are made up of three components. There is an image processor to manipulate and store data, a display to convert electrical signals into optical emissions to be viewed by an observer, and a controller to interface between the processor and display [1]. The discovery of new technologies and the realization of new applications together drive the development of these systems.

Over the past hundred years there have been two major inflection points in the creation of electrical display systems. The first was the advent of the cathode ray tube (CRT) monitor – which brought about such devices as oscilloscopes, televisions and computer monitors. Gradual screen size and color quality improvements with CRTs were made over time but limitations existed with respect to their cumbersome form factor and high operating voltage [2]. Individuals looking for a sleeker alternative to the standard CRTs drove the second inflection point in electrical display system design. With the creation of the integrated circuit and new research in the field of semiconductors, flat panel displays were created by using a multitude of thin film techniques such as plasma deposition, sputtering, photolithography, and etching [2]. Eventually, flat panel displays not only improved upon the existing CRT television and computer markets, but also vastly expanded new segments, such as the portable electronics sector, which became a multi-billion dollar market.

One downside to the current commercially available flat panel displays is their rigidity. Manufacturing constraints require glass to be used as the thin-film substrate due to its high thermal resistances and bonding capabilities. If an alternative flexible display were available however, newer market segments have growth potential where conformable and unbreakable displays are a necessity.

There are research ventures currently ongoing, such as Samsung's YOUM, which is a plastic based AMOLED flexible display technology. The techniques used to fabricate YOUM prototypes are reportedly very slow and expensive compared to their glass counterparts [3].

This report will detail the design and analysis of an alternative flexible electrical display system based on existing LED technology for niche low cost applications like those of the emerging wearable electronics market. An integrated hardware-software portable display system is developed to give users the capability to display digital content on a flexible 224-pixel, 10cm x 15cm 24-bit color screen. Ultimately, research in the flexible display space with advancements from projects such as this will hopefully bring about a third inflection point in the display system market – with the creation of innovative products covering totally new applications that CRTs and rigid flat screens could not do by themselves.

Chapter 1: *System Design Overview*

1.1 Background

To build any complete electrical display system, multiple hardware and software components are required. On the software side, different layers are needed to allow for hardware-software abstraction [4]. This abstraction will allow a typical user to forgo the details of the hardware and keep a focus on product specific usage, which boils down to providing new content for the display. Along with this software stack, a hardware system needs to be created. For the system to be mobile it must be lightweight and it also must have a portable power supply. These two constraints, along with a price limit, set limitations on overall system performance.

The critical hardware components can be categorized into two sub-systems: the display and the control system. Both of these two sub-systems draw a finite amount of current when in use. Tradeoffs need to be made when selecting components that make up these sub-systems. An example of these tradeoffs is illustrated when deciding the screen pixel count. A display with a high number of pixels will increase the resolution, but it will also increase the current draw of this sub-system. Although the picture quality goes up, the mobility of the system goes down in that more charge is required to power the extra pixels. More current draw equates to either a lower mean time between charges (MTBC) or a system having a larger power supply to allow for equal MTBC. You can see how one system parameter, the screen resolution, sets limitations on other key parameters (in this case the power supply, system weight, and cost).

Similarly, consider a microcontroller (MCU), which is one possible component of the control sub-system. There are high-level tradeoffs when selecting a MCU in terms of cost and performance. The goal is to select a variety that will allow for system functionality while minimizing the unit price, power consumption, area/weight, and complexity. Determining an adequate variety requires a detailed understanding of how the MCU fits into the entire system – both on the hardware and software side. Understanding how the MCU interfaces with the display tells just how fast and

computationally intensive it needs to be. Also, an understanding of the software stack needs to be known to select an appropriate MCU. Part of the software stack will be running on it and tradeoffs come into play when selecting an MCU with different program/data memory sizes. There is no need for selecting an MCU with a 64KB FLASH program memory, for example, if the maximum required for the software is only half that. This will only drive up cost and power consumption unnecessarily. Also microcontrollers have their own instruction set architectures and C compilers depending on the vendor. There are complexity and schedule tradeoffs when choosing a MCU in this respect as well.

In the end, every component that makes up each subsystem has such tradeoffs. Some of the tradeoffs, like those listed above, may be obvious from the beginning of the design phase. Others may not be so obvious initially – and surface during system integration. The bottom line though is to understand how each component impacts the key design parameters of the system, and balance them accordingly to meet or exceed the initial specifications of the project.

1.2 Flexible Display Target Specifications

Any embedded system can in general be measured by their dependability, energy efficiency, weight, cost, and run time efficiency [5]. For an electronic display, there are more specific parameters, in addition to these, that ultimately quantify the device's characteristics. The following section documents the nine key design parameters shown in Figure 1 used to build the flexible display system relevant here. Initial target specifications were determined by balancing all known component tradeoffs with the features forecasted to be worthy of a device for the emerging wearable electronics market.

Design Parameter	Target Specification
Cost	50.00\$
System Weight	200g
Display Resolution	224 pixels
Display Size	10cm x 15cm
Display Aspect Ratio	1:1.5
MTBC	8 hours
Image Count	4 pictures
GMA	5 functions
Project Schedule	3 months

Figure 1: System Level Design Parameters and Targets

The first parameter was cost. If a product derivative were to be sold to a consumer using this prototype as a reference design, higher material costs would eat into any possible profits – so this parameter carried significant weight. Conscious efforts were made at each stage of the design process to select and create components with this metric in mind. The components with high price points of this system are lightweight, high energy density power supplies and LED pixel elements.

Again, the conceptualized use of the flexible display was for embedded applications. The system could be easily adapted into the emerging wearable electronics sector, for instance, as a means for users to display digital content on clothing as shown in Figure 2. One necessity of such a device is portability. To quantify this target an overall system weight specification was created to keep any product derivatives from being bulky.



Figure 2: Flexible Display Concept Application

The display size, resolution, and aspect ratio were again all chosen with the embedded wearable electronics sector in mind. A 10cm x 15cm sized flexible display would fit squarely on an adult's upper chest as part of a shirt, for example. The aspect ratio was chosen to meet that of a traditional television or computer monitor screen. A lower resolution target was more of a technical constraint based on the number of individual LEDs physically possible to pack into the allotted space. On one side it is more cost and power advantageous compared to a high-resolution screen, but it does limit the image content the user is capable to share. When considering the application as a wearable electronics device however, the low-resolution quantization effects of the display are mitigated by the fact that the viewers will be far away from the image content.

MTBC is set by the charge storage capacity of the supply and how much charge the system draws over time. This particular system's instantaneous current draw is highly variable and fluctuates over time depending on what image content is displayed on the screen. A target of eight hours was set as the mean time between charges based on

predicted device usage. Usage models will be needed to accurately predict average current, so that an appropriate power supply can be chosen to meet the MTBC constraint.

The image count is a quantifier of the functionality of the device. The idea is to allow the user to both create and upload four different images to be available for display on the screen by using a GUI software application. Once uploaded, the user can switch between image0, image1, image2, and image3 whenever he or she wants to display new content.

Another functional specification is to enable global motion algorithms (GBA) as defined in Figure 2. Instead of just having static content shown on the flexible display, this feature allows for a more entertaining and eye-catching device. The specification targets five different operation modes the user can switch between. The idea is for the user to toggle among the versions of an image that are static, flashing, upward scrolling, downward scrolling, and rightward scrolling. Graphical representations of the five global motion algorithms are shown in Figure 3 for a simple 16-pixel test case of a multi-color southeastern pointing arrow. Over time each function impacts the displayed image differently, and those impacts are highlighted over three discrete time units for demonstration.

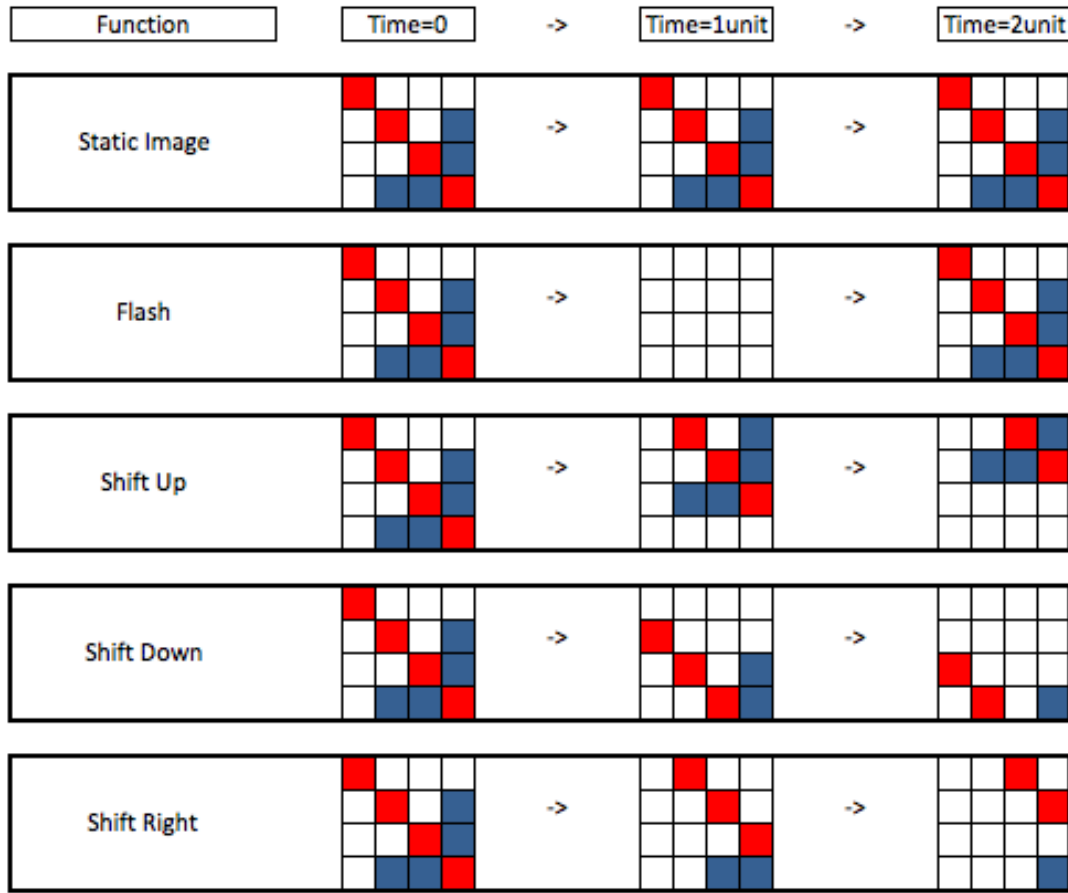


Figure 3: Global Motion Examples

Finally the last system constraint is to meet the previous eight constraints all within a limited time frame of three months. Typical product designs need to meet strict time-to-market windows to beat competitors and to catch critical consumer buying periods.

Now that the outline of a flexible display system is characterized with the needs of a wearable electronics consumer in mind, the following chapters will cover the hardware and software specific details. After that, results are shown to highlight system power efficiency, cost, and other key parameters against the initial targets outlined in Figure 1.

Chapter 2: *Hardware Design*



Figure 4: Hardware System Integration Overview

2.1 Hardware Architecture Overview

Figure 4 has two parallel images. The top picture is of the system showing 'TEST' text with two underlines in blue and yellow. The second is the same image overlaid with the key hardware components outlined in blue.

The first major component to review is the display itself. It is a mesh of RGB LEDs. Each LED has the capability to emit its own color independent of its neighbors. This is accomplished through the use of a WS2811 chipset, which is Worldsemi's LED driver circuit. Each LED has its own WS2811 to read digitally encoded color values and change the LED color accordingly. After looking at multiple vendors an integrated RGB LED + WS2811 component was selected particularly for achieving the highest display resolution possible. The operating voltage and power consumption were also rated the lowest of the different options available, which made it viable for embedded applications.

There were two main challenges with the display. First was the implementation of the actual flexible-mesh arrangement. The second challenge was empirically finding an adequate display voltage/current balance to maximize power supply life while maintaining display functionality and adequate brightness.

The breadboard in the upper right of Figure 4 holds all the control sub-system components. These components jointly take real-time input from the user to change the image on the display and also to perform image manipulations through coded global motion algorithms on the microcontroller.

The main design decision with respect to the control sub-system came down to picking a microcontroller that would meet the timing protocols of the WS2811 chipsets while minimizing cost, power, and complexity. I had previous experience with embedded C programming using the Atmel AVR – so that was a major plus over other PIC, Intel, or TI varieties. Of the low power 8-bit Atmel microcontrollers available, the ATmega382P was selected. Among the 48, 88, 168, and 328 varieties shown in Figure 5, the 328 has the most programmable flash memory (32K) and the most internal SRAM (2KB) while having similar operating voltage and frequency ranges with no significant price differential between devices [6].

Device	Flash	RAM
ATmega48P	4K Bytes	0.5K Bytes
ATmega88P	8K Bytes	1K Bytes
ATmega168P	16K Bytes	1K Bytes
ATmega328P	32K Bytes	2K Bytes

Figure 5: Atmel Device Memory Comparison

For no major cost or complexity increase, the 382P variety provides a larger RAM, which would allow for more manipulation of screen images. In terms of performance, the ATmega328P can also operate anywhere between 4MHz and 20MHz depending on the operating voltage. In this system, the operating voltage of the MCU is set by the maximum frequency needed in order to meet the timing requirements of the WS2811 chipset. Figure 6 shows this relationship. More details on the WS2811 handshake protocols are covered in Sections 2.3. The software written to follow these protocols is covered in Section 3.4.

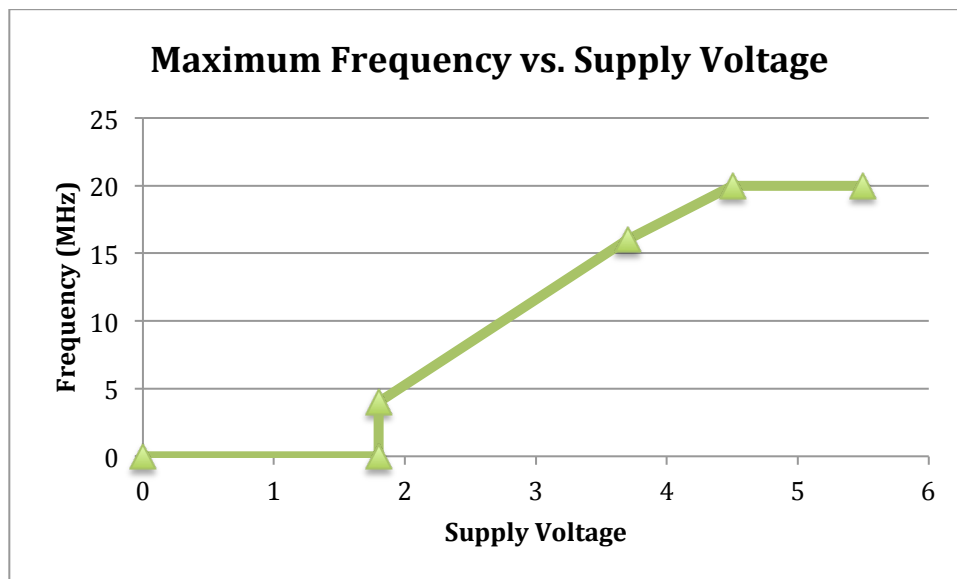


Figure 6: ATmega328P Maximum Frequency vs. Supply Voltage

The lower right breadboard in Figure 4 holds the power delivery sub-system. There is a power supply consisting of two single-cell LiPoly batteries and a microUSB port to allow for system re-charging. Like many other embedded systems the challenge with the power delivery subsystem was creating an infrastructure that would support the peak current draw of the display and control sub-systems, as well as providing adequate MTBC as designated in the target specifications.

2.2 Flexible Display Construction

The flexible mesh display is made up of two different components – RGB LEDs and WS2811 chipsets. The WS2811s are daisy chained together to refresh the display depending on input from the microcontroller [7]. Each WS2811 controls an individual RGB LED pixel. A sixteen-pixel example of the logical connection of this is shown in Figure 7.

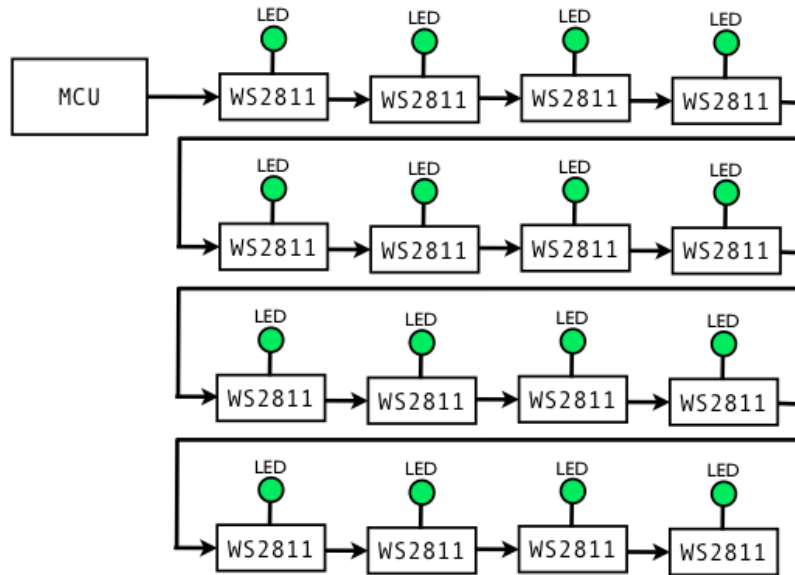


Figure 7: WS2811 Logical Connectivity Example

Reels of RGB LEDs were purchased that have each WS2811 built-in from international supplier JiaHuiYuan Industry located in Shenzhen City, China. Each reel

had a four-meter strand containing 60 LEDs per meter. The strand was 1cm wide and was a flat-flexible wire with the LEDs and WS2811 ICs soldered in series as depicted in the following figure. Figure 8 shows the arrangement of LEDs, WS2811 ICs, and the dataflow direction (DIN to DOUT) to communicate with the microcontroller. The reader should note the finite space between individual LED+WS2811 combinations. This allows cutting and splicing from the main stand to create segments of smaller length.

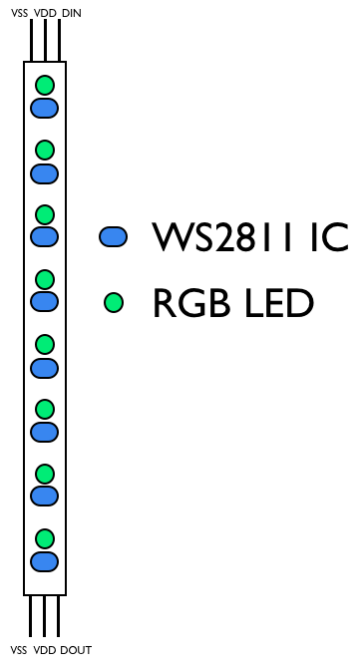


Figure 8: Integrated LED+WS2811 Reel Segment

To create the flexible display, small segments such as the one shown in Figure 8 were horizontally abutted to form Figure 9. There are two things to notice from this figure. One is the flipped direction of data flow for each LED column. The data flow for the left most column (column 0) is down. The data comes in from the MCU at the top (DIN pin) and travels through to the DOUT pin at the bottom of column 0. The adjacent flipping of data flow for each column allows for concise daisy chain connections. The DOUT of column 0 feeds the DIN of column 1 (second LED column from the left). Without column flipping, the data signal would need to be continuously routed across the

re-wiring and soldering and would not be ideal. It would remove the display flexibility component around the horizontal axis (solder joints are not flexible under normal operating temperatures). It also wouldn't completely fix the issue with horizontal LED alignment. There would be improvement – but it would not be perfect. Solder was also found to be unreliable in flexible electrical applications such as this due to its brittle nature. All of these reasons drove for an alternative solution.

This alternative solution required a method of LED/WS2811 folding, which all but eliminated the unwanted space between vertical LEDs. By taking advantage of the extra wiring between LED+WS2811 groupings, a series of joints were constructed in the flexible strip. This kept the electrical connectivity sound and also effectively folded the WS2811 chipsets underneath the LEDs.

Figure 10 shows a side view of an example strand using this new folding technique. The black line is the connecting flexible wire that has the set folding points, which allow for the WS2811s to slide underneath the LEDs. The blue blocks designate the newly hidden WS2811 chipsets from the viewer's point of view. This ultimately allows for the shadow area of the strip to be completely consumed by the LEDs themselves, effectively eliminating the white space between them.

Folding each strand in such a fashion did require manual crimping, but is significantly less complicated compared to cutting, re-splicing, and soldering connections to perform vertical display compression. It is also more reliable.

From Figure 10 you can see that the LEDs are no longer all flat against the original reference surface. Instead they are angled due to having the WS2811s slid underneath them. Because the height of the WS2811s is small however, the new LED pitch does not introduce any distortion when looking at the images generated on a display having this vertical compression technique. It was also possible to keep the alternating column data flow direction, for concise data signal daisy chaining between LED columns.



Figure 10: Integrated LED+WS2811 Segment Side View

An example of the final screen after taking a reel of RGB LED+WS2811 chipsets, cutting the total strand into columns, horizontally abutting alternating columns, and vertically compressing each column looks like the display shown in Figure 11.

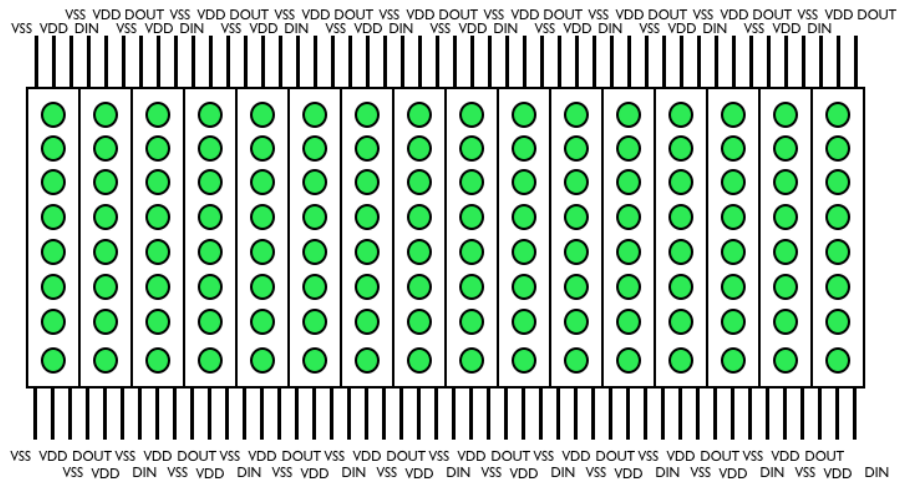


Figure 11 – Integrated LED+WS2811 Folded Technique, Top View

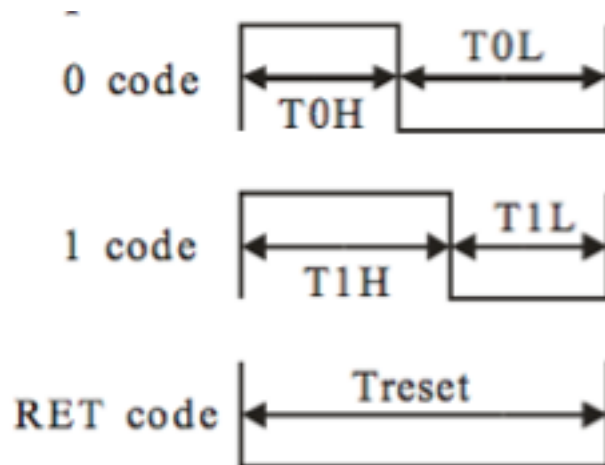
One drawback to the reel provided by JiaHuiYuan Industry was that the entire four-meter strand was not continuous. There were solder joints every 30 LEDs (every 50cm). The solder connections would shift the horizontal LED row continuity if they were in the design. To alleviate this, the display height was set to 14 pixels (2x14 columns come out of each continuous strand without the solder ball connections). This set the display height at 10cm after vertical compression. If the pixel height was set greater than this, for example, to 16 pixels, only one 16 pixel segment would be taken out

of the 30 pixel segment between solder joints. This would leave 14 unused pixels. The wasted material would drive up overall material costs.

2.3 LED Pixel Controller

The WS2811 chip takes in 24 bits of data and generates 3 pulse-width modulated signals each to control the red, green, and blue color of the RGB LED it controls [7]. As shown previously, they are designed with the intent to be daisy chained together with a central control system supplying the color refresh data. To do that, each WS2811 first takes the initial 24 data bits and latches them internally. The WS2811 holds these values until it receives a reset sequence. If no reset sequence is recognized and more data comes on the data input, the WS2811 does not process this information and change the LED color, but instead sends it downstream to the next WS2811 after doing some pulse reshaping to alleviate any distortion from traversing through the WS2811 daisy chain.

Each bit of data is transferred asynchronously to the first WS2811 in the chain. The WS2811 interprets a '1' or a '0' based on the pulse widths of the incoming data. The WS2811 can operate in either high speed or low speed mode. The chosen mode of operation impacts the zero and one codes understood by the WS2811. This mode can be configured in the DIP version of the WS2811, but not in the integrated version used to create the flexible display. The integrated WS2811s were pre-wired in the high-speed mode – meaning the zero and one pulse-width codes are twice as fast and follow the specifications shown in Figure 12 [7].



Name	Description	Duration	Slack
T0H	0 code, high voltage time	0.25us	+/- 75ns
T0L	0 code, low voltage time	1us	+/- 75ns
T1H	1 code, high voltage time	0.6us	+/- 75ns
T1L	1 code, low voltage time	0.65us	+/- 75ns
Reset	low voltage time	> 50us	

Figure 12: WS2811 Handshaking Protocols

To reiterate, each WS2811 reads 24 bits of data encoded as shown above. Each 24-bit data packet contains three eight-bit values for the red, green, and blue color values to set the LED controlled by the WS2811. There is a particular order to this data packet. It is first the green byte, then red byte, followed by the blue byte. For each byte, the WS2811 interprets the first bit received as the most significant bit of the byte packet.

Details on the software written to handle the handshaking protocols of the WS2811s along with measured pulse widths for the zero and one codes are covered in Section 3.4.

2.4 Power Delivery

The minimum frequency the MCU needs to operate is 16MHz. This is set by the tight timing requirements of the WS2811 bit codes mentioned in Section 2.3. Following the voltage vs. maximum frequency curve in Figure 6, the MCU needs to run at or above 3.7V [6]. The LEDs themselves are rated for 5V. Empirically however, they were found to be functional at 3.7V with no yield issues in terms of LED luminosity or WS2811 data capture. This simplified the need for a shared supply for both the control and display sub-systems in that there was no step-up circuitry required.

3.7V LiPoly batteries are commercially available and come in a multitude of different capacities. They have the advantage of being rechargeable, lightweight, and capable of high energy densities. To meet the current demands of the flexible display, two LiPoly batteries were used: one for supplying the even display columns plus the control sub-systems and the other for supplying the odd display columns. This ensured no supply current saturation occurred under any operating conditions.

Section 4.2 covers the analysis of the power delivery network and how the battery capacities were determined based on predicted usage models.

Chapter 3: *Software Design*

3.1 Software Architecture Overview

Figure 13 shows the hardware/software partitioning for the flexible display system. There are multiple layers of abstraction in this design as described in the background material in Section 1.1. There is a Visual Basic Application, which allows users to create images for the flexible display system without knowing anything about the hardware or lower level software components under the hood. There is also C code that gets compiled to run on the microcontroller, which is burned along with the user image pixel data, that has procedures to handle user input along with a device driver procedure for the WS2811 chips. The device driver takes care of interfacing with the WS2811 chipsets daisy-chained in the flexible display. All of these layers together provide a complete software package for the embedded system.

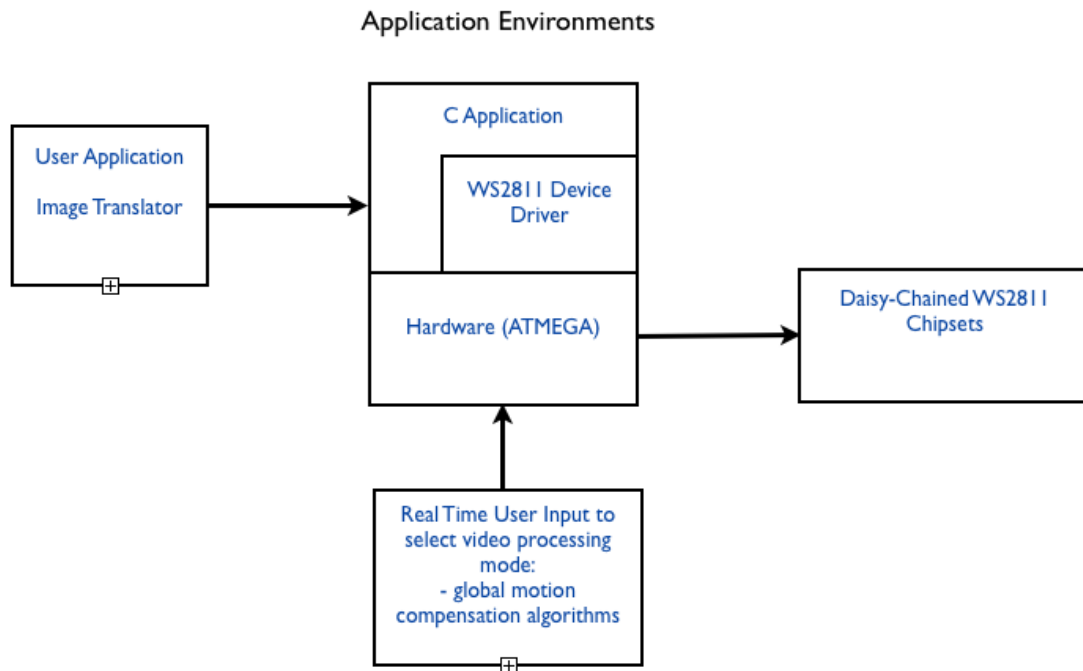


Figure 13: System Software Interaction

3.2 Image Translator User Application

The goal of the image translator is to provide the user with a framework to manipulate the flexible display without having to know any of the hardware or any of the underlying software components. This was accomplished by taking advantage of the graphical nature of Microsoft's Visual Basic language. The Visual Basic application created gives the user the ability to create four different images that will be burned onto the microcontroller. A screenshot of the User Application is shown in Figure 14 with four sample images.

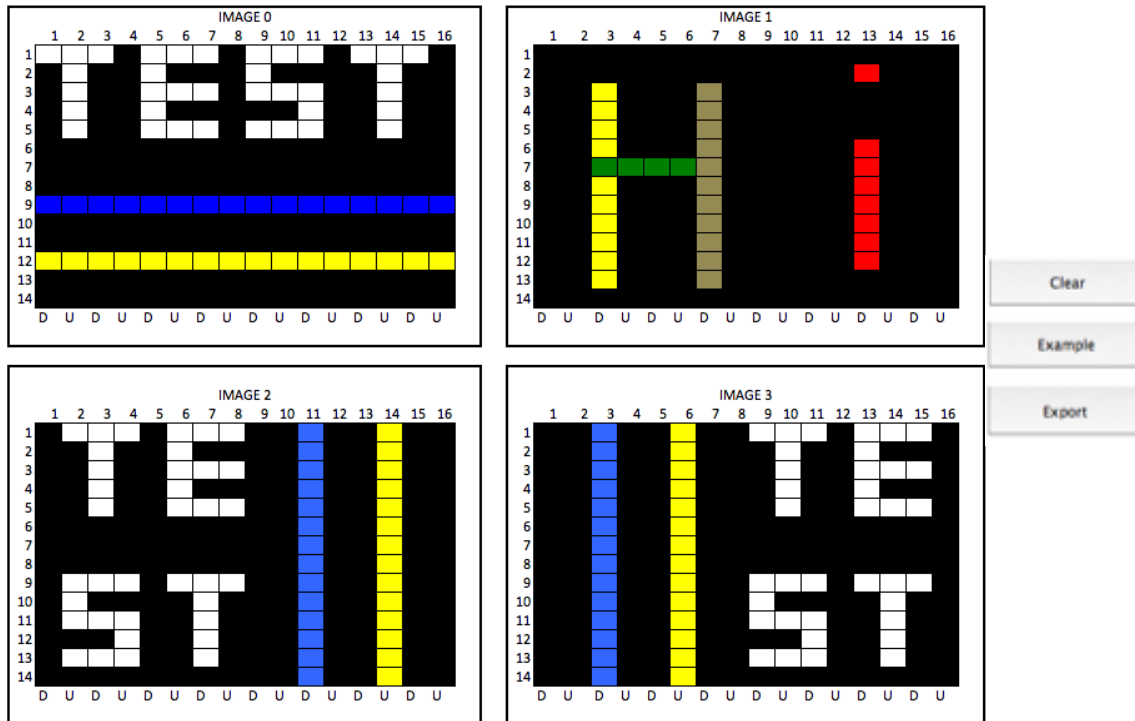


Figure 14: VBA Image Translator User Application

This application is an embedded Excel Visual Basic Application. The user is able to use the Excel fill tool to easily manipulate each cell's color to create different images he or she would like to see on the flexible display.

After image creation is complete, the user clicks the **Export** button. When activated, this command calls a Visual Basic macro `get_rgb`. This macro reads the color of each cell in the images created by the user and decomposes the red/green/blue color triplet of each. The values for each cell are written to a file `pixel.rgb.data` containing an array of image data to be read by the C code compiled for the microcontroller.

The `get_rgb` Visual Basic procedure is capable to handle different screen sizes and different image counts. Currently the code is set to handle 4 user images each of which has 224 pixels (16x14). Parameterization techniques were used to ensure easy portability for any hardware modifications.

The `get_rgb` procedure explicitly stores all image data in FLASH memory. This is designated by the `PROGMEM` keyword [8]. Because each image is 224 pixels, that equates to 2688 bytes of user image data per burn:

$$\left(224 \frac{\text{pixels}}{\text{image}}\right) \times \left(1 \frac{\text{byte}}{\text{color}}\right) \times (3 \text{ colors}) \times (4 \text{ images}) = 2688 \text{ bytes}$$

It was not possible to store the image data in SRAM or RF because the total SRAM/RF storage capacity is only 2KB with the ATmega328P. There is EEPROM storage available as well but is limited to only 1KB, so the 32KB FLASH was used. The downside of saving the images in the FLASH program memory and not the SRAM data memory is that they can't be changed between microcontroller burns. One benefit, though, is that more than four images can easily be added. The current size of the boot loader, system control software, and device driver takes 4KB of flash. That leaves enough space for a total of 41 images.

To pull all of the image color content, the macro loops over each image and steps through each cell [9]. The stepping across each image is not random – the order is set depending on the WS2811 daisy-chain order as described in Chapter 2. The daisy-chain configuration can be interpreted from the “D” and “U” keys at the bottom of each image column in Figure 14. These keys show the direction of data flow when refreshing the flexible display and whether the data flow is down or up for a given column.

The cell color extraction happens by accessing the cell object's interior color attribute [10]. The red/green/blue breakdown for each cell can then be gleaned and is saved to the pixel data file. Finally, a message box lets the user know the `get_rgb` procedure completed and the export was successful.

For more details on the Image Translator User Application please refer to the Appendix for the actual code.

3.3 System Control Software

The open-source Arduino IDE was used to write the system control software to be compiled for the microcontroller. Altogether this code handled I/O transactions, interfaced with the user application, and called the WS2811 device driver assembly program. This code, `flex_display.pde`, has two high level routines: `setup` and `loop`. The `setup` procedure is executed only once on MCU reset and has necessary I/O pin configuration settings and variable initializations [8]. The `loop` procedure is executed repeatedly and contains the main functionality of `flex_display.pde`.

The `loop` procedure first does user input polling. As previously mentioned in Chapter 1, the user has the ability to select one of four images to display as well as a mode of how to display it. These two user inputs are first checked at the beginning of the `loop` procedure. Pin 9 and 10 control the user image selection and image function, respectively.

The user has the ability to continuously change either the image or the mode. When a change is detected on the input pins, variables are incremented accordingly. For a change in the image selection, a variable named `offset` is incremented. The variable is used to change where in program memory to look for the new image. For a change in the image mode, a variable named `function` is incremented. Both `offset` and `function` are incremented up until a certain number of maximum images and functions (4 images and 5 functions) before being rotated back to 0. After this polling and variable assignment is complete, a switch statement is entered which calls a different image mode procedure depending on the value of `function`.

The variable `function` can take on five different values. The mapping between each value and the corresponding procedure executed is:

1. `hold_picture`
2. `flash_picture`
3. `scroll_picture_down`
4. `scroll_picture_up`
5. `scroll_picture_right`

Each different sub-procedure calls the device driver that executes a screen refresh. The screen refresh device driver is given an array of pixel color data (3 bytes for each pixel). This pixel data is read from program memory using the `pgm_read_byte_near()` function. The four images are stored in program memory in a particular order: `image0`, `image1`, `image2`, followed by `image3` as shown in Figure 15. Depending on the mode and memory offset, an array is filled with the pertinent color data to be given to the WS2811 device driver.

The code to perform image motion was more challenging. Again the code did not alter the actual images in program memory but re-arranged how the image content was read. The order of pixel information was a function of the mode selected by the user and required different reading patterns depending on pixel location and screen column data flow. For more details please see the `flex_display.pde` code in the Appendix.

raw_pixel_data_FLASH	2687	image3	blue[223]
			green[223]
			red[223]
			...
			...
			...
			...
			blue[0]
			green[0]
	2016	image2	red[0]
	2015		blue[223]
			green[223]
			red[223]
			...
			...
			...
			...
			blue[0]
			green[0]
	1344	image1	red[0]
	1343		blue[223]
			green[223]
			red[223]
			...
			...
			...
			...
			blue[0]
			green[0]
	672	image0	red[0]
	671		blue[223]
			green[223]
			red[223]
			...
			...
			...
			...
			blue[0]
			green[0]
	0		red[0]

Address Offset

Figure 15: Memory Map of Image Content

3.4. Display Device Driver

Refreshing the screen with new content was not handled in C or the Arduino IDE. This is because the timing constraints of the WS2811, as described in Figure 12, required each bit of information to be written in 1.25us. There are built in C Arduino functions that are capable to measure time and stall program execution, but not at the granularity necessary to correctly write to the WS2811 chipsets (`delayMicroseconds` is accurate only for 3us and up) [11]. Therefore a custom Atmel AVR assembly procedure `REFRESH_SCREEN` was written to handle the handshaking with the WS2811s. This is possible by writing a C wrapper around an assembly procedure [12]. The WS2811 protocol expects 24 bits of information for each pixel in the following order:

Green[7:0], Red[7:0], Blue[7:0]

`REFRESH_SCREEN` therefore takes in the pertinent pixel color byte information (as a pointer to the first element of an array), and then loops through each green bit, each red bit, and then each blue bit for a total of 24 iterations for each pixel. Figure 16 shows the assembly code and data flow for the first green loop.

Each instruction executes in 1 cycle (cycle time = 62.5ns). To write a ‘1’ to the screen, the MCU output pin (Port B pin 0, or `PB[0]` was used) needs to send a 625ns pulse followed by 625ns of `PB[0] = 0`. This is shown in the Bit=1 column. To write a ‘0’ to the screen the timing was tighter – and ended up defining the maximum frequency needed to run the microcontroller. The MCU needs to output a 250us pulse in this case. These timings are shown in the left half of Figure 16.

	Cycle	Time (us)	Bit=0	Bit=1	Branch1	Branch2	Branch3
GREEN LOOP	1	0.0625			PB[0] 0 to 1		
	2	0.125			is green[7] == 1?		
	3	0.1875			if YES goto Branch2		
	4	0.25			PB[0] 1 to 0	nop	
	5	0.3125			nop	nop	
	6	0.375			nop	nop	
	7	0.4375			nop	nop	
	8	0.5			nop	nop	
	9	0.5625			nop	nop	
	10	0.625			nop	PB[0] 1 to 0	
	11	0.6875			goto Branch3	goto Branch3	
	12	0.75					nop
	13	0.8125					nop
	14	0.875					nop
	15	0.9375					nop
	16	1					nop
	17	1.0625					bit_count--
	18	1.125					left shift green[7:0]
	19	1.1875					is bit_count > 0?
	20	1.25					if YES goto Branch1

DARK GREEN = Pin HIGH
 LIGHT GREEN = Pin LOW

Figure 16: REFRESH_SCREEN Byte Loop Pseudo Code and Timing

The pseudo-code to accurately generate the correct pulse duration is shown on the right half. No matter what bit is being written, the PB[0] output pin is first sent high. Then, the most significant bit of the green color byte is compared to see if it is a 1 or a 0. This was effectively accomplished with the CPI instruction by comparing the entire green byte to the immediate value 128. If the green byte is greater than or equal to 128, then the MSB of the green byte is known to be 1 (for an unsigned comparison). During the CPI instruction execution, a status register (SREG) is updated with the compare results [13]. These comparison results are then used by the upcoming branch instruction, which will change program flow depending on whether the MSB is a 0 or a 1. This was accomplished doing an unsigned comparison using the BRSH instruction.

If the most significant bit of the green byte (green[7]) is one, the program counter is then moved to the first line of Branch2, which extends the PB[0] pulse duration. If green[7] is 0, then the PB[0] output is quickly transitioned back to 0 to meet the 250ns

pulse duration requirement. At the end of both Branch1 and Branch2, the 0 and 1 pulses have been written. Code execution then jumps to Branch3.

After appropriate delay, Branch3 decrements a register that holds a current bit number variable. Initially this register (r20) is given the immediate value of 8 (for the number of bits in a byte). After each loop iteration, `bit_count` decrements to keep track of how many bits remain to be written for each byte. Because the initial comparison in Branch 1 is done to the most significant bit, the next most significant bit is left shifted into the most significant bit position. Finally, a comparison is done. If the `bit_count` variable stored in register r20 is 0, then the loop has gone through all eight bits of the green byte. In this case execution stops looping through Figure 16 and moves on to the red byte loop.

The red loop and the blue loop are identical to that shown in Figure 16. The only difference is that for the blue loop the last bit is unrolled. This allows for cycles 12 through cycles 16 to be used to pull in the next pixel's green, red, and blue bytes instead of just as nop stall instructions.

Figure 17 shows oscilloscope waveform measurements of a series of '0' pulses being written to the display. Vertical cursors show that the pulse width durations fall within WS2811 specifications by 65ns.

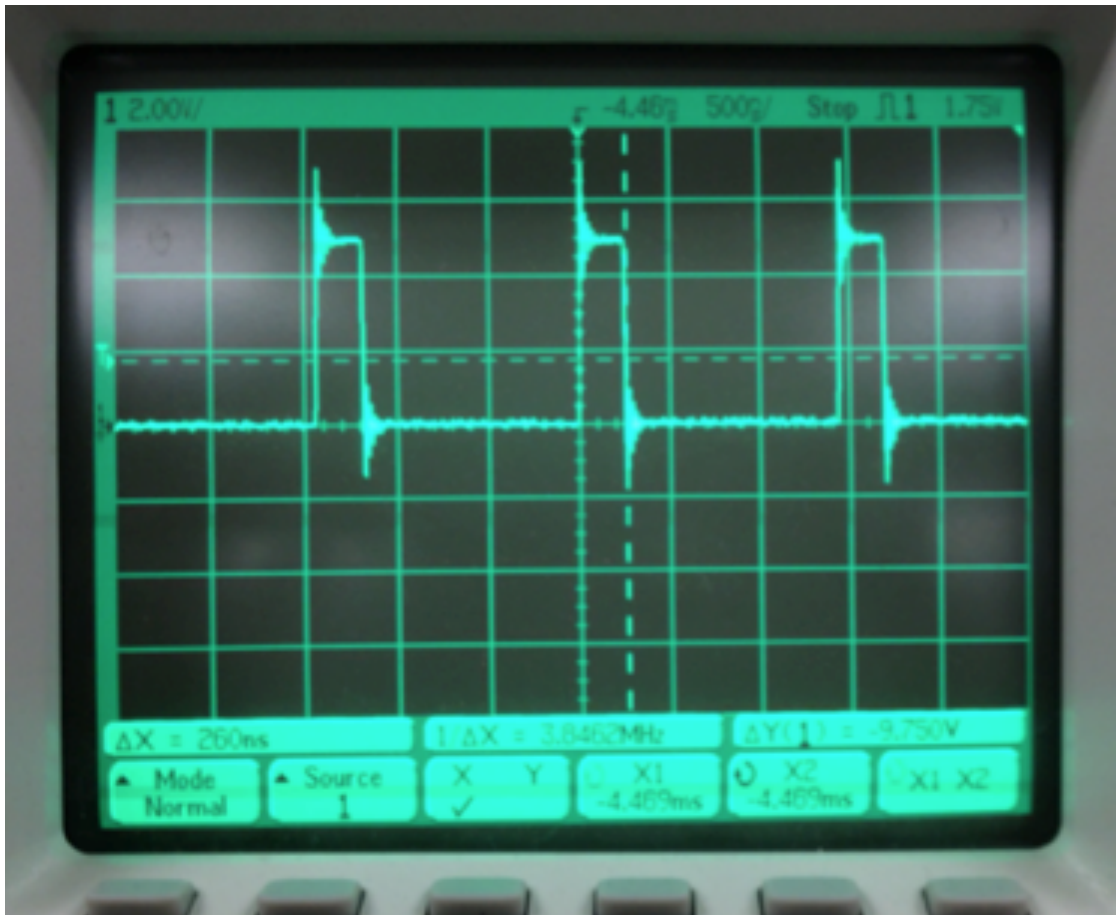


Figure 17: Scope Readings of WS2811 '0' Codes

Similarly Figure 18 shows oscilloscope waveform measurements of series of '1' pulses being written to the display. In this case vertical cursors show that the pulse width durations fall within WS2811 specifications by 45ns.

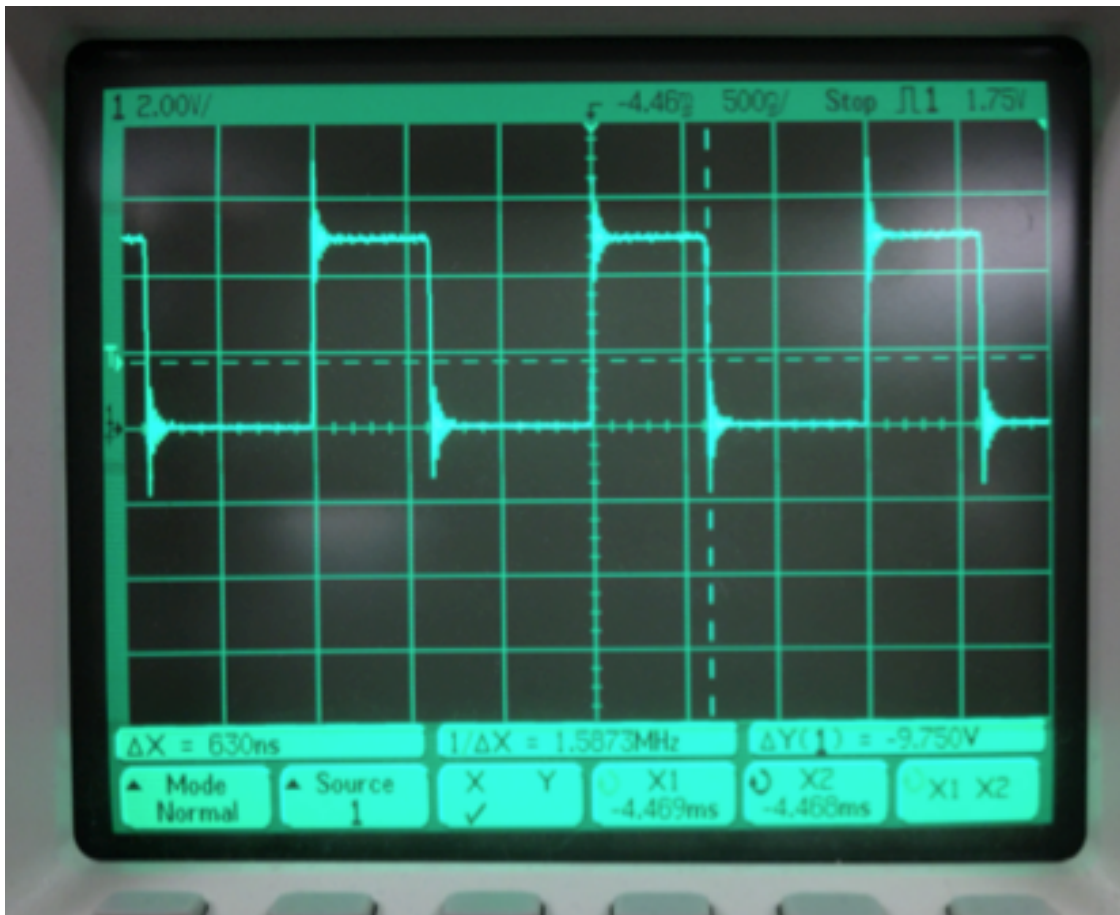


Figure 18: Scope Readings of WS2811 '1' Codes

Chapter 4: Results

4.1 Power Analysis

To meet the weight and MTBC targets, single cell 3.7V Lithium Ion rechargeable batteries were used as the power source for this embedded system. The batteries are very lightweight, have a high energy density, and have a compact form factor. Electrical component vendors carry numerous different capacities that range from several hundred mAh to upwards of 4000mAh. For the fixed voltage, the total average system current draw for the device using these batteries can be described as:

$$I_{system_{avg}} = I_{screen_{avg}} + I_{control_{avg}}$$

The screen current designates the component of the total system current consumed by the flexible display and the control current designates the component of the total system current consumed by the microcontroller and peripheral IO interfaces for user interaction. In order to meet the initial design specifications set in Figure 1 of eight hours between power supply charges, models are needed to accurately characterize this system current over different operating conditions. Such models are needed because of the many different states of operation of the system. Once these models are created, the power supply infrastructure can be extrapolated to meet the peak current and average current demands of the system. Together, these are the overall goals of the system power analysis.

As outlined previously in Chapter 1, there are five modes of operation possible for the flexible display. The user can generate a static image, a flashing image, an image that scrolls up, an image that scrolls down, and an image that scrolls right. Also, the user has the capability to customize each of the 224 pixels in the display with 24-bit RGB color. Altogether that makes:

$$5 \text{ modes} \times 224 \text{ pixels} \times 2^{24} \text{ colors} \approx 18.8 \times 10^9 \text{ distinct system states}$$

Measuring current usages of the device at almost nineteen billion states to completely characterize the system is impractical given schedule constraints. A more realistic approach is to estimate the upper and lower extremes of system current consumption

based on pixel color, screen usage (number of pixels on), and operation mode. This will create an envelope encompassing all system states to adequately profile projected usage to allow for accurate power estimates.

Gaining knowledge of how color impacts this current envelope will greatly simplify the solution space of the problem by removing a variable with 2^{24} possible values. The RGB color model is based on the superposition of the three primary colors – red, green, and blue. Each discrete 24-bit color value is made up of some intensity of these three colors. For example, black is made of 0% red, 0% green, and 0% blue whereas white is made up of 100% red, 100% green, and 100% blue. Measuring the total system current while the display content is varied with respect to these different color combinations will provide a color current-draw envelope.

This was accomplished by using the test images shown in Figure 19, which have the following 24-bit hexadecimal representations:

Black - 0x000000

Green - 0xFF0000

Red - 0x00FF00

Blue - 0x0000FF

White - 0xFFFFFFFF

These four images keep the pixel-on count constant at 56 (0.25x total screen usage), but vary the emitted pixel color among red, green, blue, black, and white.

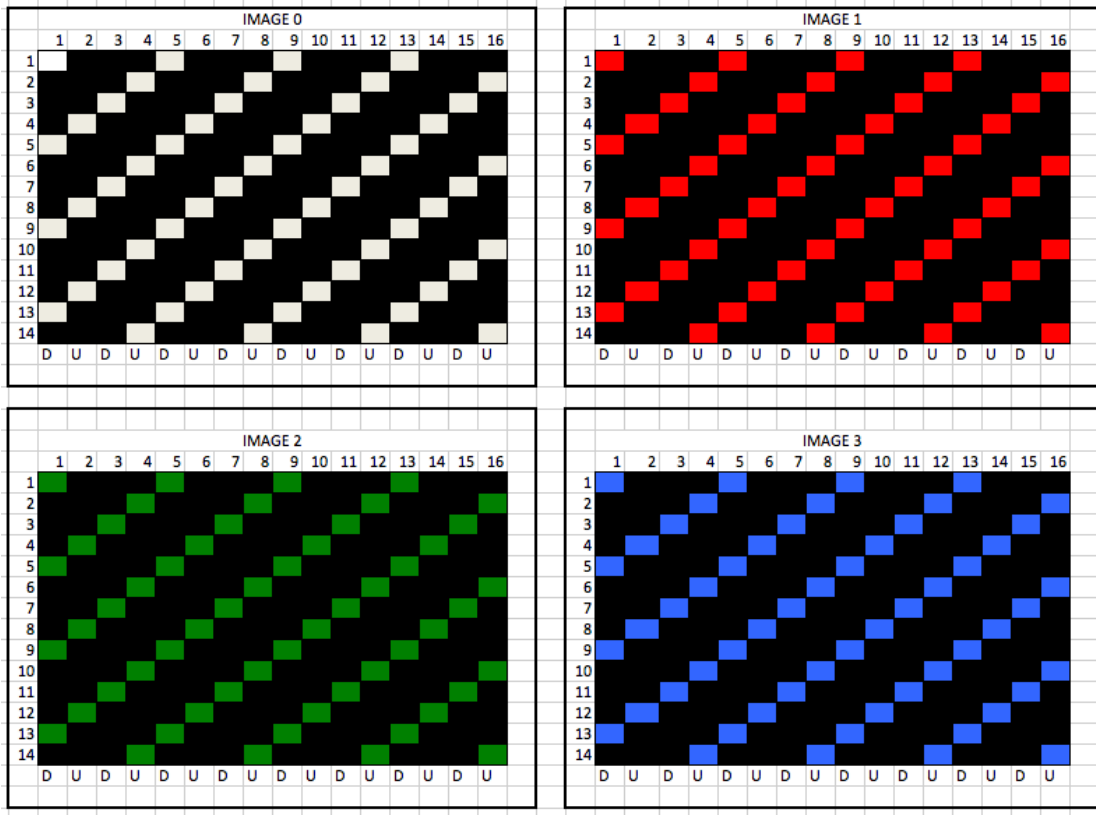


Figure 19: 0.25x Pixel Usage Color Variant Images

Figure 20 shows the relationship between color and user mode for a fixed number of pixels on (56). As expected, this shows how the total system current is highest for white light, which is additive of the three primary colors. It also shows that for displaying only a black screen the total system current is the least. These two colors define the 24-bit color envelope because all 2^{24} possible colors fall within these bounds. For reference, solid green, red, and blue are also provided and they fall within the bounding curves of the white and black light. For system current measurement data please refer to the power analysis backup data in the appendix.

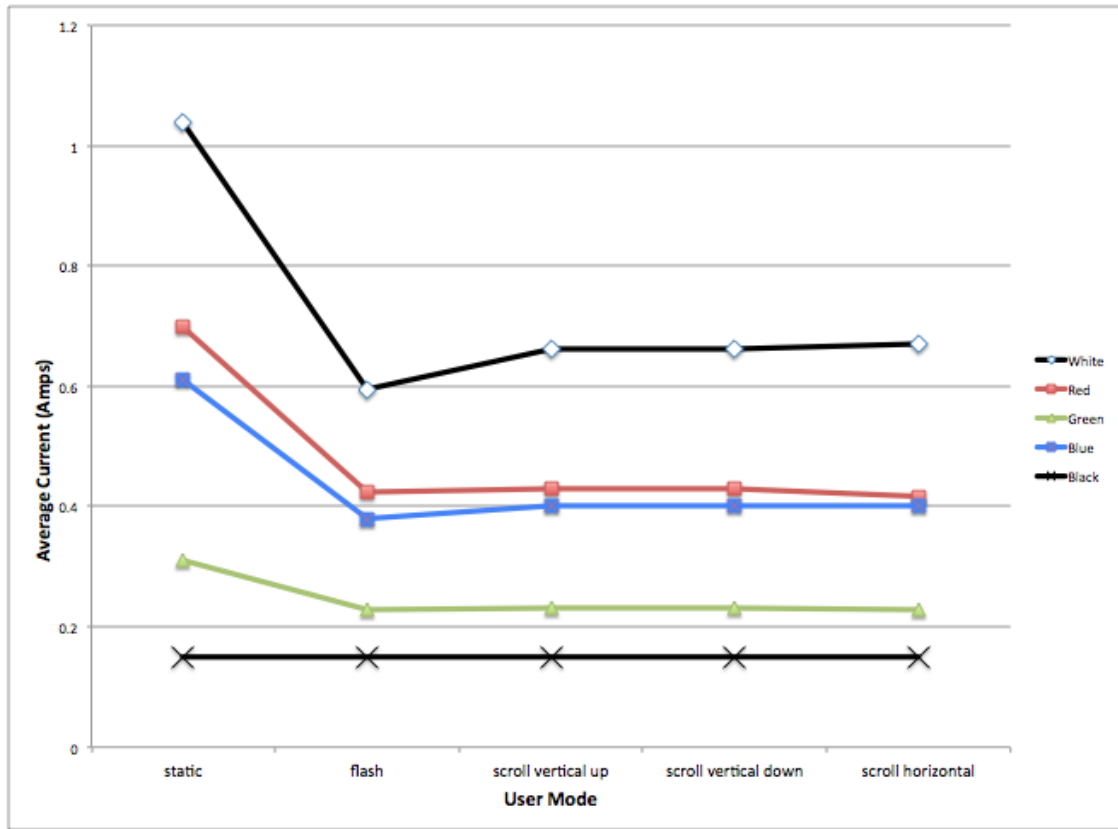


Figure 20: Average System Current verses User Mode with 0.25x Display Usage

To further quantify the current envelope in terms of user mode, pixel color, and quantity, it is necessary to see how the total system current changes with respect to pixel quantity. To get an accurate upper bound of the current envelope across all possible operating conditions, the four images in Figure 21 were used. By using only white light, an upper bound to the current envelope of the system is found. This bound can be interpolated between 4 sample points across all possible 224 pixel count – 0.25x screen usage, 0.5x screen usage, 0.75x screen usage, and 1.00x screen usage.

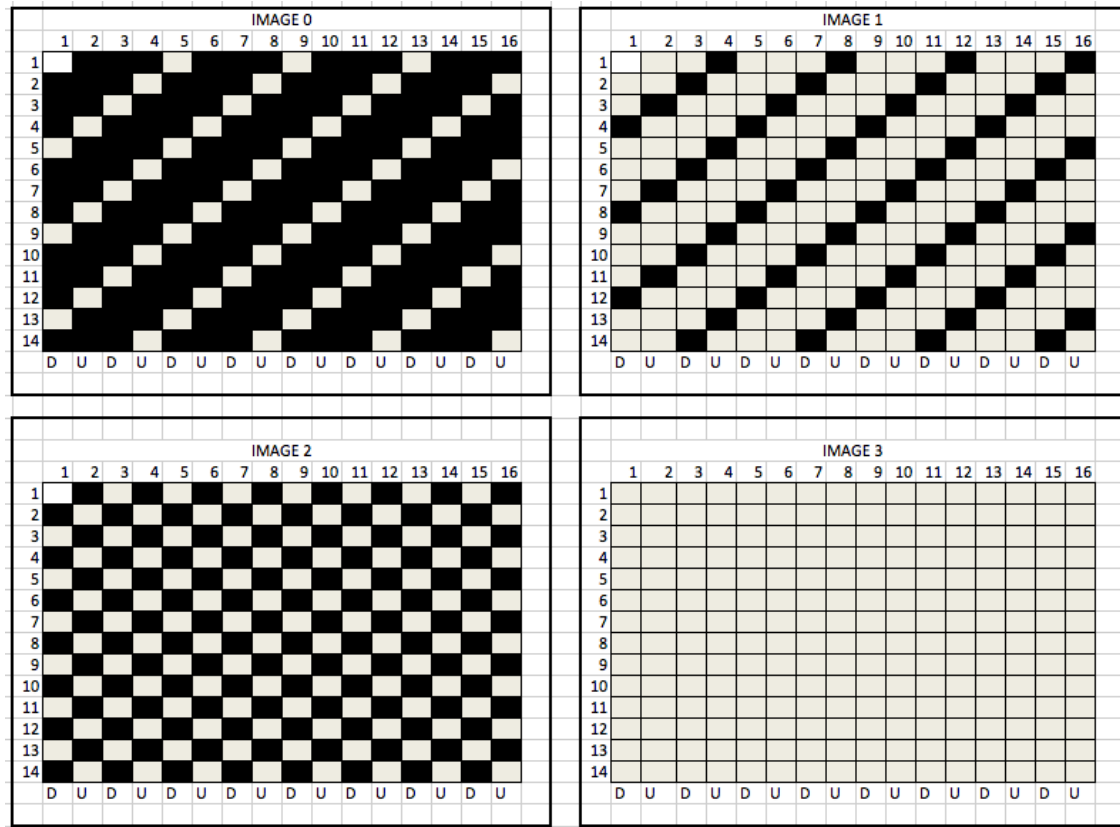


Figure 21: Images Used to Determine an Upper Bound on Total System Current

Figure 22 shows the relationship between pixel count variation and total system current for each of the five operating conditions. As expected, the total current increases for more pixels emitting light. The static image consumption is also the highest current consumer compared to the other modes. Flashing an image on and off averages both the high current consumed during the static image display and also that of the low current draw when the screen is completely off. The other modes have this same effect where a fraction of the entire image is shown at certain periods in time – reducing the overall average current over a finite usage period. Please refer to the Appendix for current measurement data with respect to the images shown in Figure 21.

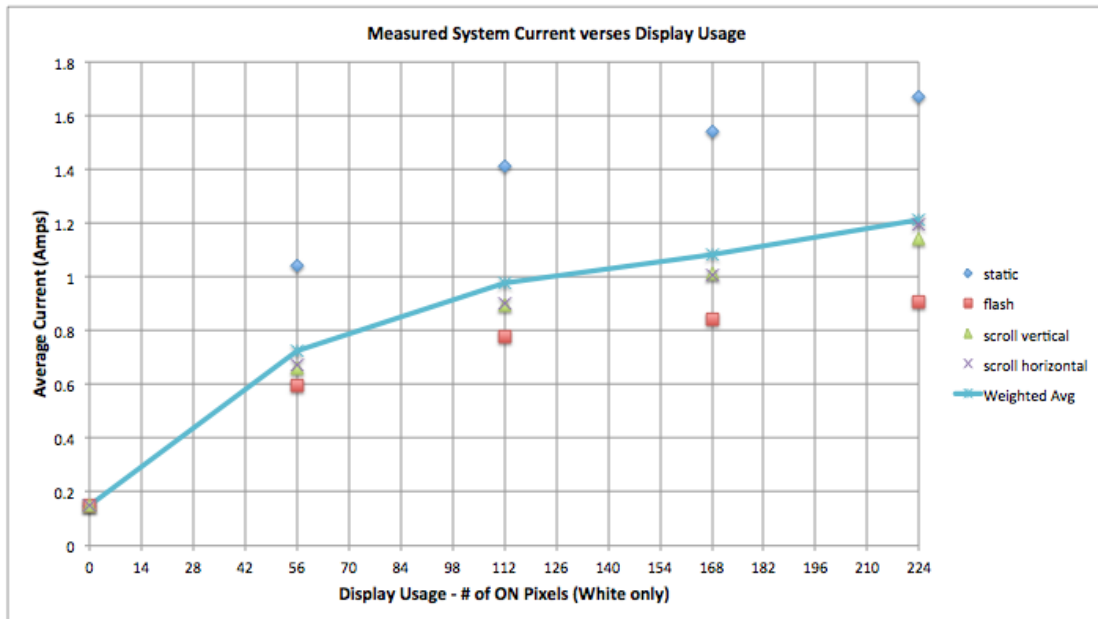


Figure 22: System Current versus Pixel Count (White Light Only)

Assuming equal usage times among all five modes, a weighted average curve was generated in Figure 22 as the upper bound estimate for the system current envelope. Again, this assumes over any period of time, the user will activate all five modes equally. The lower bound was extracted in a similar way – by measuring the system current of the device while the screen was completely black. Across all modes the only current is then that of the microcontroller and idle current draw of the WS2811s, which collectively drew a steady 150mA. Together, these two bounds on the device current-draw enclose all possible system states based on predicted usage. They are shown in Figure 23.

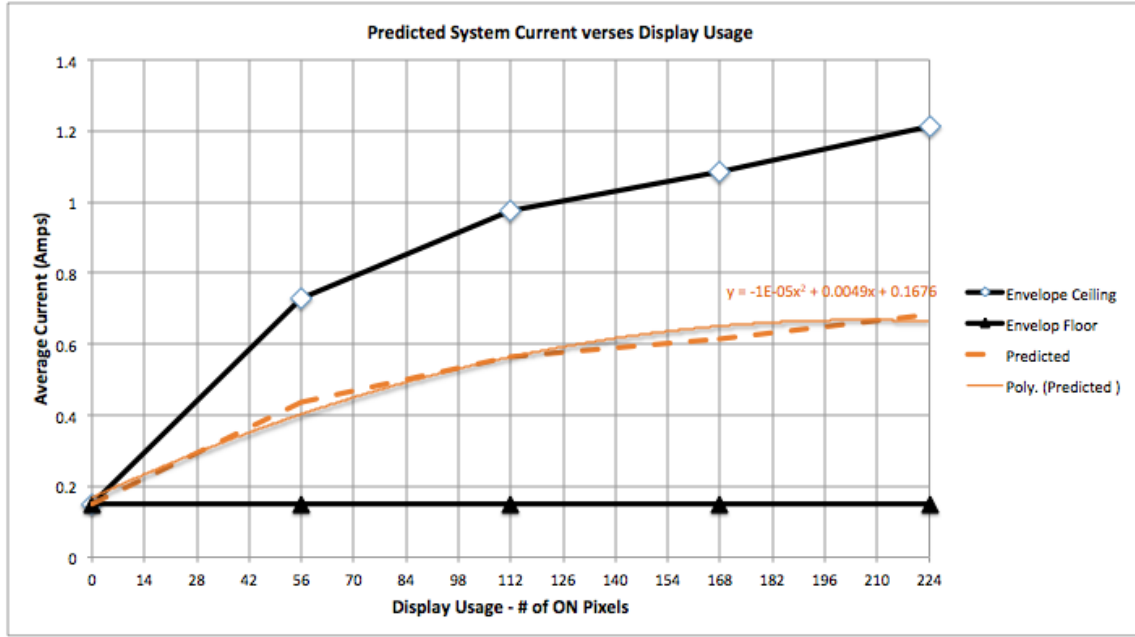


Figure 23: System Current Envelope versus Pixel Count

Under normal operation then, it is assumed a user would be in the middle of this current envelope, which is designated by the dotted orange line in Figure 23. That would be because, on average, the image would be made up of multiple colors and not just black and white, which produce the derived current envelope for the system. A polynomial was fit to the predicted current usage data points, which interpolates the current usage per pixel from the derived 0.25x, 0.5x, 0.75x, and 1.0x values. The total current draw averaged over all modes and colors can then be expressed as a function of just the pixel count:

$$I_{average} (Ampere) = 10^{-5}p^2 \times 0.0049p \times 0.1676$$

Typical usage suggests that a realistic pixel-on count per image to be around 64. Sample 64-pixel images are shown in Figure 24. That would imply from the equation above that the predicted current draw of the system is 440mA. To meet the initial specification of a mean time between recharges of eight hours, a power supply with at least 3520mAh charge capacity is required. Figure 24 shows the actual implementation of the example

images generated for Figure 25. A 4000mAh supply was chosen due to the finite granularity in battery supply increments.

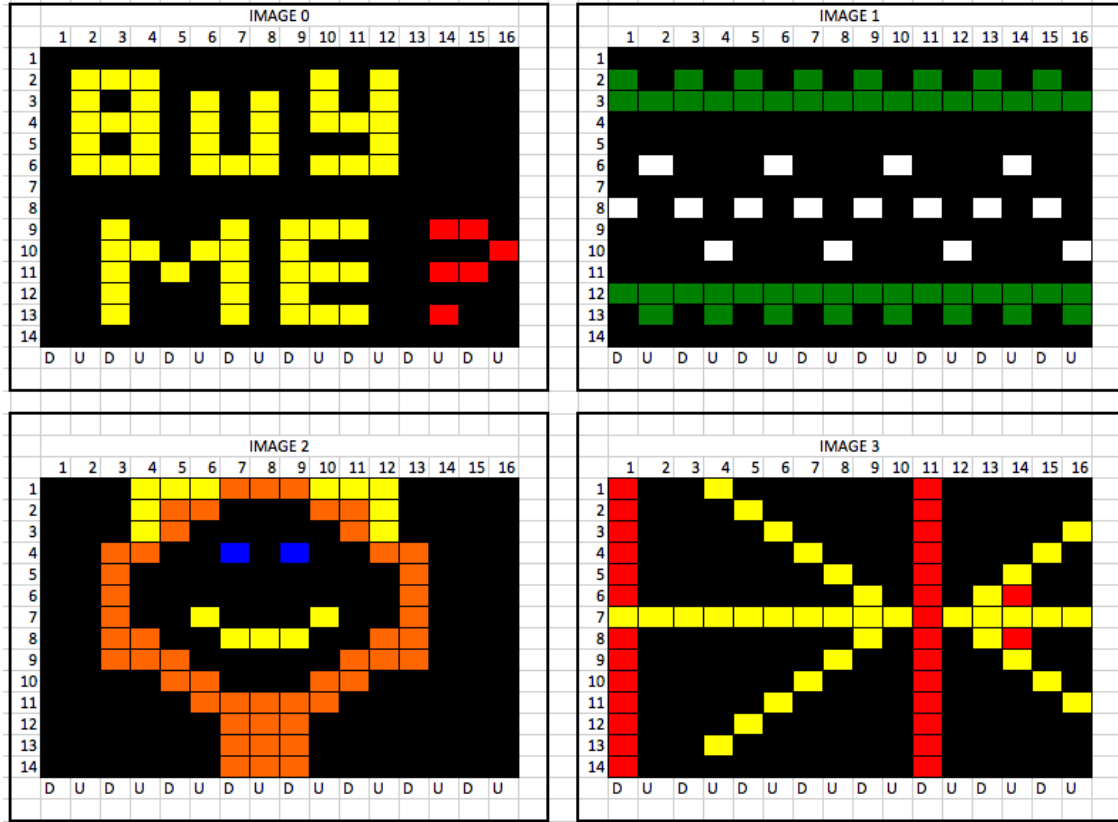


Figure 24: Example 64-pixel Images

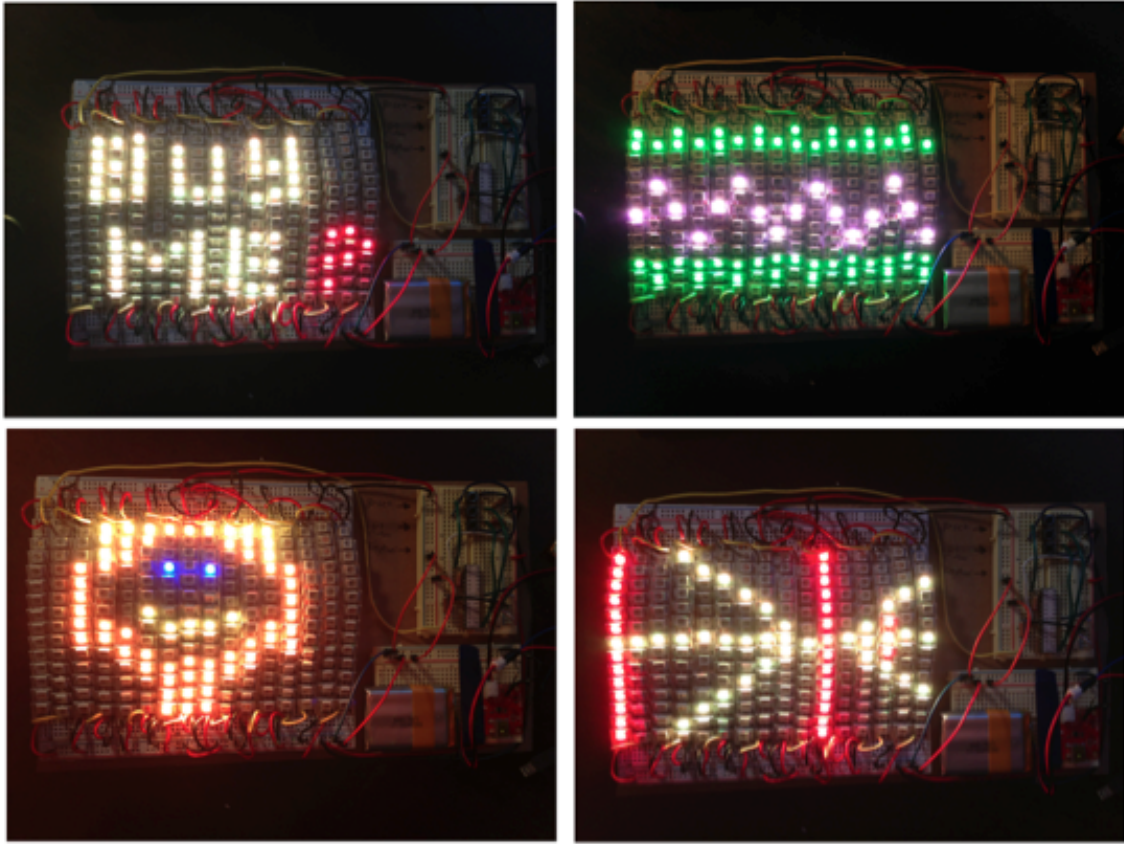


Figure 25: Example 64-pixel Image Implementations

4.2 Power Supply Infrastructure

In addition to maximizing the mean time between charges, the screen integrity was also a concern. For images that require greater than 1A of current, there is a noticeable, uniform, color distortion across the screen when using just a single cell supply. For example, with a single system supply powering a static 100% white screen as shown in the bottom right of Figure 21, not enough current is sourced causing whites to appear red instead.

From the data sheets available on the LiPoly batteries, the maximum discharge current is 2.0 times the capacity of the battery [14]. So according to the data sheet, no matter if a 1000mAh or 2000mAh cell was used the system current needs should be met (that is a 2A and 4A max discharge current respectively). Empirically however the

findings did not match. The maximum discharge current achievable was 1A per cell – which was the source of the color degradation.

To alleviate this, two smaller single-cells were used instead of one large single-cell. One 2000mAh cell was used to power half of the screen columns. The second 2000mAh cell was used to power the remaining screen columns along with the control hardware. This resolved the distortion issues seen when high current images were displayed.

4.3 Cost Analysis

Figure 26 shows the breakdown by individual component of the total cost to build the flexible display system, which came just over \$80. There are three heavy hitters that dominate the price point of this device. First are the integrated RGB LED/WS2811 segments that make up the display. From the beginning this was a known fixed cost. The only possible relief with these are vendor discounts when buying in bulk.

The other two dominant components that rule the overall system cost are the power supply and the power supply charger. The cost of the power supply more than doubled at the tail end of the project design cycle in order to fix the color distortion issue discussed in Section 4.2. The solution for this unfortunately required two relatively non-standard 3.7V 2000mAh LiPoly single cell batteries instead of one 3.7V 4000mAh LiPoly single cell battery. A single 4000mAh battery costs \$9.30 whereas two 2000mAh batteries cumulatively cost \$21.98. To charge these batteries a third-party Li-Poly charge management controller was used. Cumulatively, these three components make up almost 95% of the total device cost.

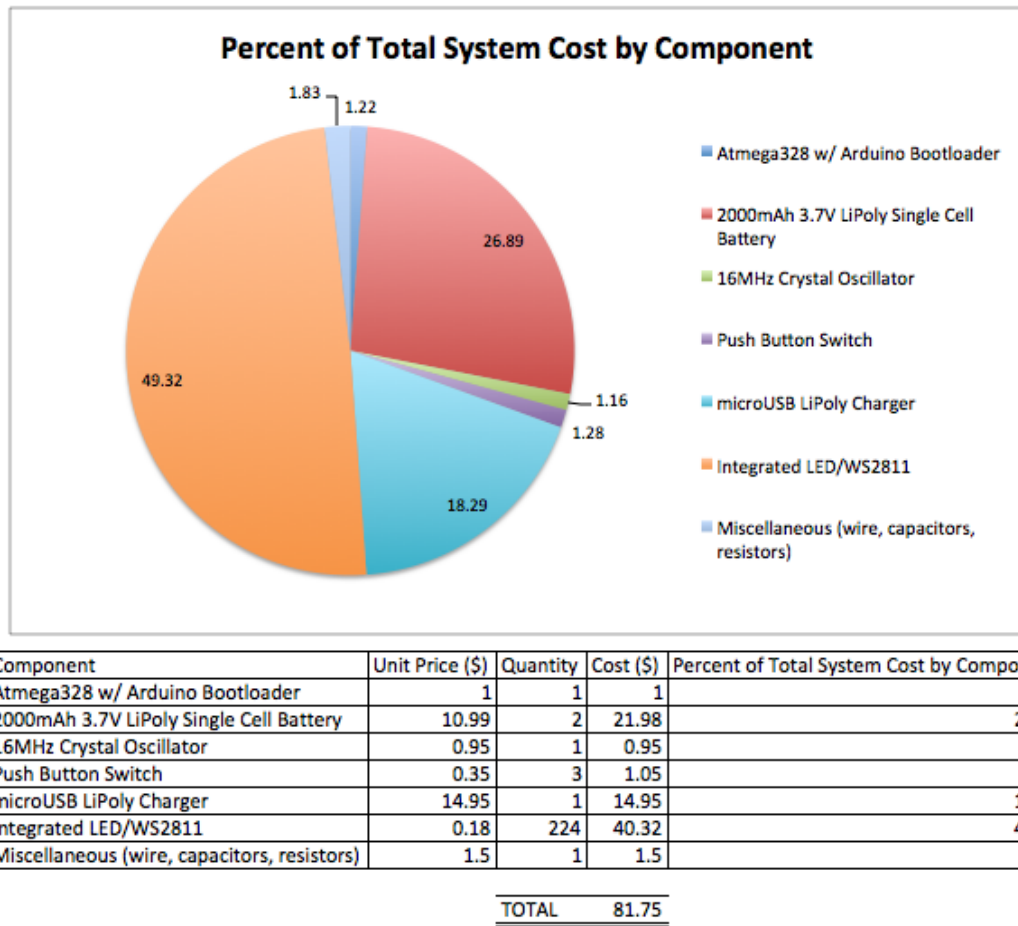
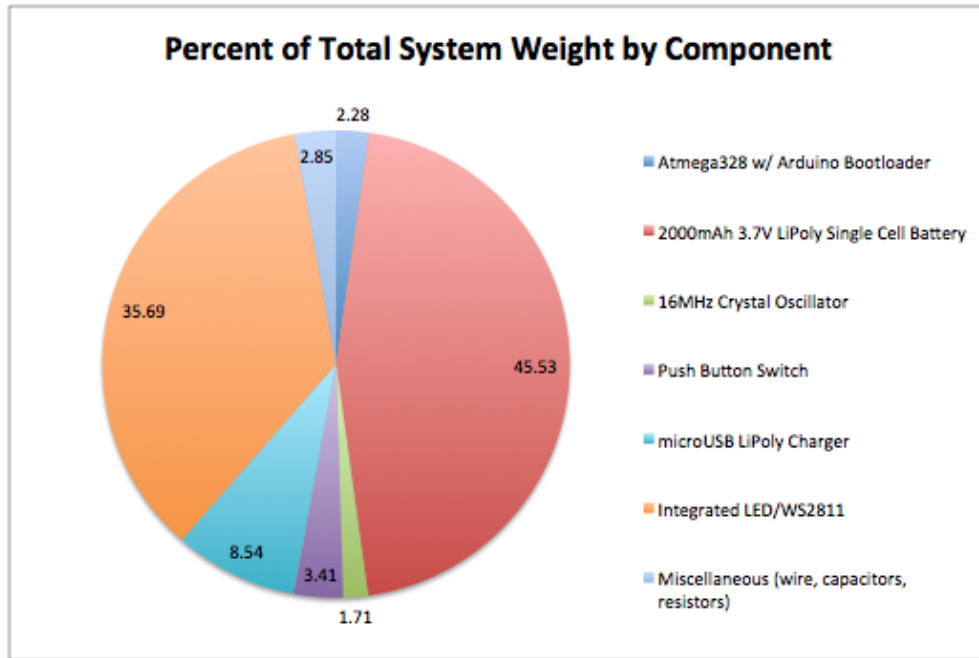


Figure 26: Overall System Component Material Cost

4.4 Weight Analysis

Figure 27 shows the breakdown by component of the overall measured weight. The batteries and the integrated LED/WS2811 segments make up 81.2% of the total.



Component	Quantity	Unit Weight (gm)	Weight (gm)	Percent of Total System Weight by Component
Atmega328 w/ Arduino Bootloader	1	4	4	2.28
2000mAh 3.7V LiPoly Single Cell Battery	2	40	80	45.53
16MHz Crystal Oscillator	1	3	3	1.71
Push Button Switch	3	2	6	3.41
microUSB LiPoly Charger	1	15	15	8.54
Integrated LED/WS2811	224	0.28	62.72	35.69
Miscellaneous (wire, capacitors, resistors)	1	5	5	2.85
TOTAL			175.72	

Figure 27: Overall System Component Weight

4.5 Summary of Results

Figure 28 highlights the original system target specifications outlined in Chapter 1 and the goals achieved at the end of the project. Of the nine parameters, the system cost was the only one not met. Chapter 5 will cover future design enhancements that will bring this value closer to spec.

Design Parameter	Target Specification	Actual	Specification Met?
Cost	50.00\$	81.75\$	No
System Weight	200g	175.72grams	Yes
Display Resolution	224 pixels	224 pixels	Yes
Display Size	10cm x 15cm	10cm x 15cm	Yes
Display Aspect Ratio	1:1.5	1:1.5	Yes
MTBC	8 hours	8 hours	Yes
Image Count	4 pictures	4 pictures	Yes
GMA	5 functions	5 functions	Yes
Project Schedule	3 months	3 months	Yes

Figure 28: Summary of Results

Chapter 5: *Conclusions*

5.1 Summary of Key Contributions

This report documents the design and analysis of a fully functional 224 pixel 10cm x 15cm flexible LED display system. A top to bottom software stack is shown to provide screen refresh capabilities, handle real time image manipulation, and provide a user friendly application for image creation. The total package weighs 175.72 grams and costs \$81.75.

5.2 Future Work

Given the initial constraints of the project as defined in Chapter 1, the key areas for improvement are in power performance, cost, and screen resolution.

The major power consumers of the device are the WS2811 chipsets and the LEDs. These are relatively fixed unless new integrated components are developed. There are power improvements to be made with respect to the control sub-system however. There is a power-down mode available with the Atmega328P that was not taken advantage of with this project. This mode has a rated low power current consumption of 0.1uA at 1MHz [6] compared to the measured 30mA active current draw of the microcontroller at 16MHz. In the system level software, instead of polling continuously for new user input, the microcontroller could enter this sleep mode until user input triggers an interrupt. Once an interrupt is triggered the microcontroller could enter a burst 16MHz mode to update the screen. This technique of alternating between sleep and burst modes would improve the overall power efficiency of the system by reducing the average active current of the microcontroller over time, all with only minimal software updates (1-2 weeks of work).

Reducing the overall cost of the device while keeping the same functionality would require implementing hardware sub-systems in a more price-effective way. The microUSB Li-Poly battery charger stands out as one of these that could easily be targeted for improvement – as it is the third most expensive component used to build the system at

\$14.95. It is third party IP that integrates a charging circuit with status LEDs, jumper connections, USB connections, and various mounting connections unused in this device all on a single PCB. A custom charger could be built instead using Microchip’s linear charge management controller for 0.59\$ [17].

Various techniques are described in Chapter 2 detailing the different approaches used to improve the flexible display screen resolution. Ultimately the approach used was to horizontally abut vertically folded pixel columns. Additional improvements could be made on this implementation by removing the unused space between vertical columns as shown in Figure 29. Each vertical column is 10mm wide. There is 1mm of buffer between the edge of the RGB LED and the column wire boundary as designated by the dotted red lines in Figure 29. By removing this space on the left and right sides of the LED, 2mm could be removed per column, thus increasing the overall resolution by 20%.

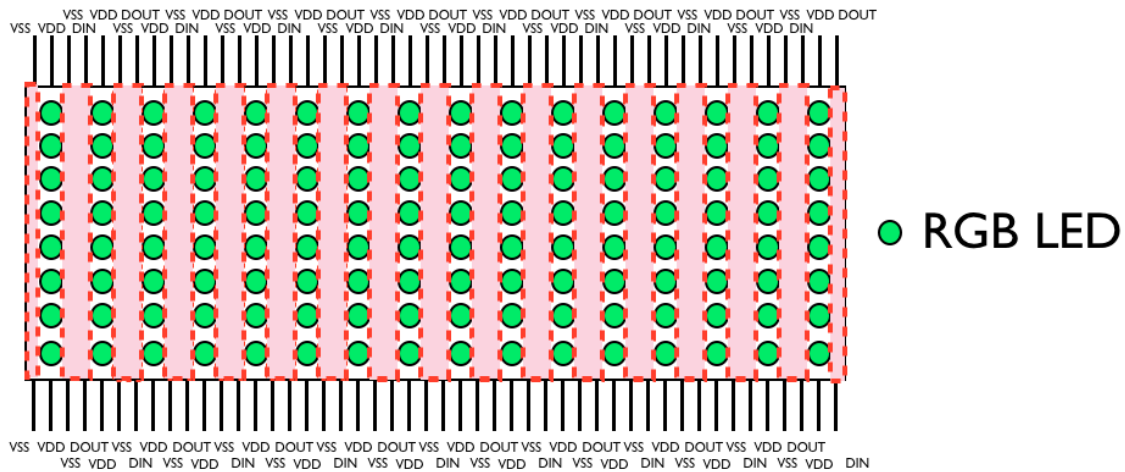


Figure 29: Flexible Display Resolution Improvement

There are multiple ways both in hardware and software to improve upon the overall functionality of the system. These ideas are highlighted below:

- Image Data Compression – Instead of storing 3 bytes per image (2688 bytes), take advantage of the fact the human eye cannot discern between the 16.77 million colors representable with the 24-bit depth different colors [18]
- Wireless Image Uploading
- Web Based Image Translator – Instead of an Excel VBA Macro, port the user application to the Internet. This feature in conjunction with a different image uploading protocol would allow the user to upload new content when desired.
- PCB prototyping – Currently the display and control sub-systems are all mounted on breadboards. Designing printed circuit boards would be the next step in product development.

5.3 Related Technologies

Many advances are actively being made with flexible electronics. The annual Flexible and Printed Electronics Conference & Exhibition highlights the top research in this field. One of the key award recipients at this year's 2013 exhibition was Plastic Logic. This company received the FLEXI R&D award for a scalable color filter alignment process, which enables the integration of color displays into flexible components [19].

When such R&D sites such as Plastic Logic partner with OEMs, innovative devices can be created and made available to the public. One such example of this collaboration is Samsung, which reportedly partnered with Vitex Systems to develop Samsung's YOUM flexible display line. Launched in early 2013, YOUM panels are supposedly to be coupled in the near future with existing smartphone designs to enlarge the device's viewable screen by having it curve around the corners of the phone [3].

There are product concepts more directly aligned with the emerging wearable electronics market, which was primarily considered when developing the target specifications for this product. A noteworthy related technology in this regard is the first programmable t-shirt, TshirtOS, by CuteCircuit. It is a 1024 pixel LED screen embedded into a cotton t-shirt that allows the wearer to display digital content [15]. In addition to

the display it reportedly has a built-in camera, microphone, speaker, and accelerometer all controllable via a smartphone. As of April 2013, there are no publicly available specifications or price listings as the design is still in the prototype stage [16].

Appendix: *Links to Code and Detailed Measurements*

Power Analysis Backup Data

The current measurements for each operating condition are provided by the author here: https://webpace.utexas.edu/mcs2955/power_analysis_backup_data.xlsx

User Application Code

The image translator embedded Excel VBA macro is downloadable and provided by the author here: https://webpace.utexas.edu/mcs2955/user_app.xlsm

System Control Software Code

The system control C code is downloadable and provided by the author here: https://webpace.utexas.edu/mcs2955/flex_display.pde

Display Device Driver Code

The device driver assembly code is downloadable and provided by the author here: https://webpace.utexas.edu/mcs2955/device_driver.h

References

- [1] Leachtenauer, Jon. Electronic Image Display: equipment selection and operation. Bellingham: SPIE – The International Society for Optical Engineering, 2004.
- [2] Hilsum, Cyril. 2010. “Flat Panel Electronic Displays: A Triumph of Physics, Chemistry, and Engineering” Retrieved April 6, 2013, from <http://rsta.royalsocietypublishing.org/content/368/1914/1027.full>
- [3] OLED-Info, Metalgrass Software. 2013. “Samsung YOUM” Retrieved April 6, 2013, from <http://www.oled-info.com/samsung-youm>
- [4] Jerraya, Ahmed. Embedded Software for SOC. Norwell: Kluwer Academic Publishers, 2003.
- [5] Marwedel, Peter. Embedded System Design. London: Spring, 2011.
- [6] Atmel Corporation. 2009. “8-bit AVR Microcontroller with 4K/8K/16/32K Bytes In-System Programmable Flash” Retrieved April 6, 2013, from <https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf>
- [7] Worldwide Semiconductor Corporation. 1999. “WS2811 Signal Line 256 Gray Level 3 Channel Constant Current LED Drive IC” Retrieved April 6, 2013, from <http://www.espruino.com/datasheets/WS2811.pdf>
- [8] Monk, Simon. Programming Arduino. New York: McGraw-Hill, 2012.
- [9] Jelen, Bill, and Tracy Syrstad. VBA and Macros for Microsoft Excel. Indianapolis: Sams Publishing, 2004.
- [10] Shepherd, Richard. Excel VBA Macro Programming. New York: McGraw-Hill, 2004.
- [11] Barnett, Larry O’Cull, and Sarah Cox. Embedded C Programming and the Atmel AVR. Clifton Park: Thomson Delmar Learning, 2007.
- [12] Kipp, Harald, 2002. “AVR-GCC Inline Assembler Cookbook” Retrieved April 6, 2013, from http://www.nongnu.org/avr-libc/user-manual/inline_asm.html
- [13] Atmel Corporation, 2010. “8-bit AVR Instruction Set” Retrieved April 6, 2013, from <http://www.atmel.com/images/doc0856.pdf>

- [14] Unionfortune Electroic Corporation, 2006. "063450 Li-Polymer Battery Packs Specification" Retrieved April 6, 2013, from <https://www.sparkfun.com/datasheets/Batteries/UnionBattery-1000mAh.pdf>
- [15] CuteCircuit, 2013. "TSHIRTOS" Retrieved April 8, 2013, from <http://cutecircuit.com/portfolio/tshirtos/>
- [16] UBM Tech, 2012. "T-Shirt OS – wearable, shareable, programmable clothing" Retrieved April 8, 2013, from <http://www.eetimes.com/electronics-blogs/the-engineering-life-around-the-web/4394061/T-Shirt-OS---wearable--shareable--programmable-clothing>
- [17] Microchip Technology Inc., 2008. "Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers" Retrieved April 8, 2013, from <http://ww1.microchip.com/downloads/en/DeviceDoc/21984e.pdf>
- [18] Myers, David G. Psychology. Michigan: Worth Publishers, 1995.
- [19] FlexTech Alliance, 2013. "FlexTech Alliance Announces 2013 FLEXI Award Winners, Recognizing Accomplishments in Flexible and Printed Electronics" Retrieved April 13, 2013, from <http://www.flexconference.org>